Exploring GAN Variants for Balancing Imbalanced Datasets

1. Problem Statement

In machine learning, classifiers are derived to minimize misclassification errors and thereby maximize predictive accuracy. The underlying assumption in most classification algorithms is that the dataset under study is balanced. However, in many real-world scenarios — such as spam detection — datasets are skewed, which causes models to overfit to the majority class and underperform on the minority class.

In this project, we explore the problem of class imbalance in a binary SMS classification task (spam vs. ham). The minority class, spam, is significantly underrepresented. We address this issue using Generative Adversarial Networks (GANs) to generate synthetic spam messages and balance the dataset, leading to more robust and fair spam classifiers.

2. Description of Dataset & Imbalance Analysis

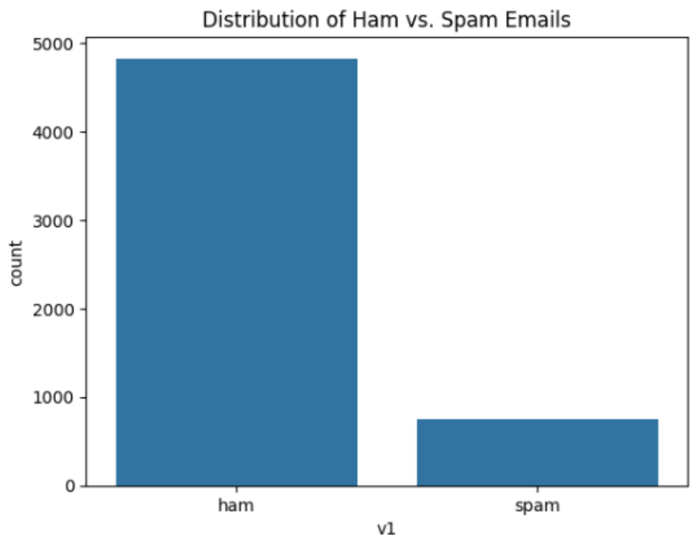
SMS Spam Collection Dataset : This dataset is sourced from kaggle:**https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset/discussion?sort=hotness**

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged acording being ham (legitimate) or spam.

The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.

The following bar chart, "Distribution of Ham vs. Spam Emails," highlights a significant imbalance between the two classes in your dataset. The "ham" (majority) class has approximately 4,825 instances, while the "spam" (minority) class has around 750 instances.

Such an imbalance can negatively affect the performance of typical classification models, causing them to favor the majority class ("ham"). Consequently, while the overall accuracy might appear high, the model may fail to accurately identify positive cases of spam, which are often the most important in a spam detection context. To address this challenge, your project focuses on exploring Generative Adversarial Networks (GANs) to generate synthetic data for the minority class, thereby balancing the dataset and evaluating the impact on classification performance.



The dataset used is a labeled SMS spam dataset with two classes:

- **Ham** (legitimate): 4825 messages
- **Spam** (unwanted): 747 messages

This yields an imbalance ratio of approximately 6.46:1 in favor of ham messages. Such an imbalance can severely hinder the classifier's ability to detect spam, the class of greater concern.

A bar chart visualized this imbalance and highlighted the need for augmentation. Only 13.41% of the samples belong to the spam class.

After preprocessing:

- **Vocabulary size**: 2194 unique tokens
- **Max sequence length** (90th percentile): 27 tokens
- **Shape of padded spam sequences**: (747, 27)
- **Batch size for training**: 64

3. Details of GAN Architectures & Training

Before training my GAN models, I first preprocessed the text data from the minority 'spam' class to prepare it for neural network input. I started by converting all the text to lowercase. Then, using regular expressions, I removed URLs, HTML tags, punctuation, numbers—basically anything that wasn't a lowercase English letter or a space.

Next, I normalized the whitespace by turning multiple spaces into single ones and trimming any extra spaces at the start or end of the text. After cleaning, I tokenized the spam texts, which gave me a vocabulary of 6,699 unique words. Each word was then converted into a sequence of integers.

To standardize the input size for the GANs, I padded the sequences based on the 90th percentile of their lengths, which turned out to be 20. Any sequence shorter than 20 got padded with zeros at the end (`padding='post'`), and longer ones were truncated from the end (`truncating='post'`).

For the embeddings, I used an `EMBEDDING_DIM` of 128, and these embeddings are learned during GAN training through a `shared_embedding_layer`. The generator's input uses a `NOISE_DIM` of 100. Finally, I created a `tf.data.Dataset` from the 747 padded spam sequences and batched them using a `BATCH_SIZE` of 64 to train efficiently.

Two GAN models were trained to generate synthetic spam messages:

1.The Vanilla GAN

Generator Architecture:

The Generator takes a 100-dimensional random noise vector as input and outputs a sequence of 20 word embeddings, each 128-dimensional. The architecture includes:

- **Input Layer:** It used to accept the noise vector.
- **Projection Layer**: A Dense layer expands the noise to 2560 units.
- **Reshape Layer**: using Reshape((20, 128)).
- **Recurrent Layers**.
- **Output Layer**: used to produce continuous embedding outputs.

Discriminator Architecture:

The Discriminator aims to classify input sequences of continuous embeddings as "real" (from the dataset) or "fake" (generated by the Generator).

- **Input Layer**: Input shape was (20, 128), same as the output of the Generator.
- **LSTM Layer**: A single LSTM(128) layer summarizes the entire sequence.
- **Dropout**: I added Dropout(0.3) to prevent overfitting.
- **Output Layer**: A Dense(1, activation='sigmoid') layer gives a probability score for whether the input is real or fake.

Training Details:

- **Loss Function:** I used for both the generator and discriminator.
- **Optimizer:** I went with the Adam optimizer for both models, setting the learning rate to **0.0002** and β_1 = 0.5.
- **Noise Vector:** The generator took in a random noise vector of size 100 .
- **Labels:** I labeled real spam sequences as 1 and fake (generated) sequences as 0.
- **Training Process:** I trained the model for 200 epochs, updating both the generator and discriminator in every step.
- **Checkpointing:** I saved the generator model with the lowest cumulative loss during training to use later for generating samples.

2.Wasserstein GAN (WGAN)

This version improves stability and reduces mode collapse by replacing the Discriminator with a Critic and using Wasserstein loss with a gradient penalty.

Generator Architecture:

The Generator architecture is exactly the same as the one used in the Vanilla GAN.

Critic Architecture:

The Critic's role is to estimate the Wasserstein distance between real and fake data distributions, rather than performing binary classification.

- **Input Layer**: (20, 128) sequence input.
- **LSTM Layer**: A single LSTM(128) layer processes the sequence.
- **Dropout**: Dropout(0.3) for regularization.
- **Output Layer**: Dense(1, activation='linear') — no sigmoid here, because the Critic needs to produce raw scores to estimate the Wasserstein distance.

Training Details:

- **Loss Functions:**
  - **Critic Loss**
  - **Generator Loss**
- **Optimizer**.
- **Noise Vector**: Same as the Vanilla GAN, I used a 100-dimensional noise vector.
- **Gradient Penalty**: Instead of clipping the weights (as in the original WGAN), I applied gradient penalty to enforce the Lipschitz constraint, following the WGAN-GP paper's recommendation.
- **Critic Updates**: For every generator update, I updated the critic 5 times to help it converge more stably.
- **Training Duration**: Just like the Vanilla GAN, I trained it for 200 epochs.

4. Classifier Setup and Evaluation

For classification, an MLP-based model with an LSTM layer was used, designed specifically for text data. The architecture comprised an Embedding layer for numerical text representations, followed by an LSTM layer (128 units), two Dropout layers (0.5 and 0.3), two Dense hidden layers (64 and 16 neurons with ReLU activation), and a single-neuron output layer with a sigmoid activation for binary classification.

The model was trained using Binary Crossentropy as the loss function and the Adam optimizer. Early stopping was implemented to prevent overfitting, with a patience of 5 epochs and a maximum of 20 epochs.

A text classifier (MLP with LSTM) was trained to distinguish spam from ham using:

1. The original imbalanced dataset
2. Dataset augmented with Vanilla GAN
3. Dataset augmented with WGAN

Classifier Architecture:

- Embedding layer followed by LSTM
- Dense layers: 64 → 16 → 1 (Sigmoid)

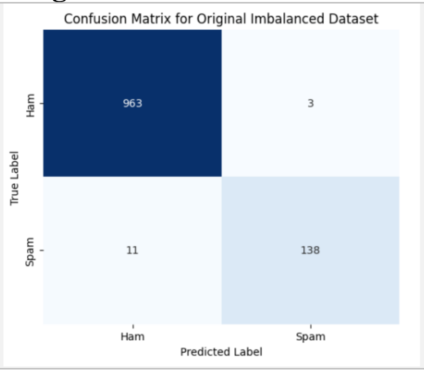
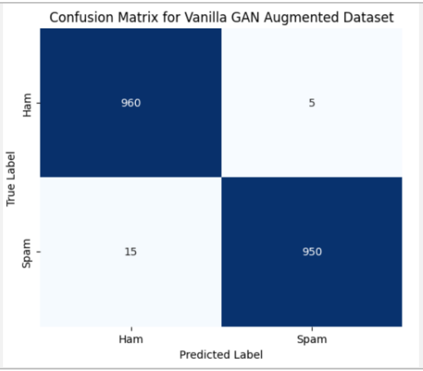
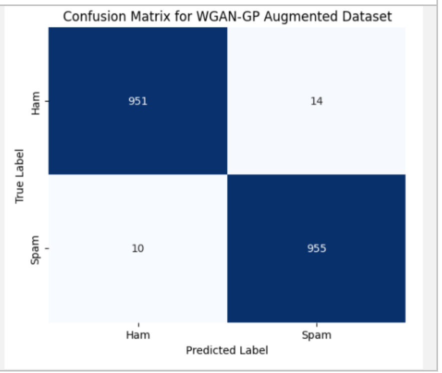
- Loss: Binary Crossentropy
- Optimizer: Adam
- Evaluation metrics: Accuracy, Precision, Recall, F1-Score, AUC-ROC

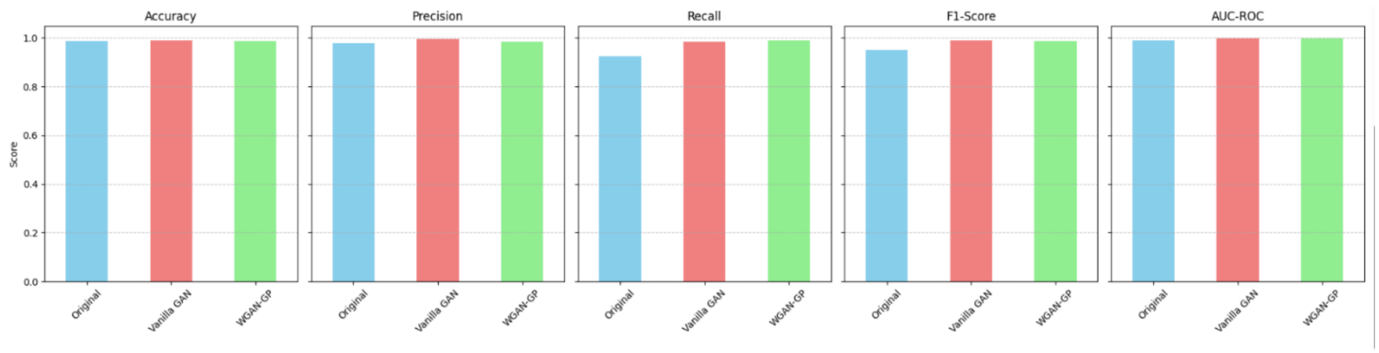
Early stopping was used to prevent overfitting, and evaluation was done on held-out test data.

Evaluation involved splitting the data into stratified training and testing sets (80/20 split). Performance was assessed using a comprehensive set of metrics: Accuracy, Precision, Recall, F1-Score (all for the 'spam' class), and AUC-ROC. Confusion matrices were also generated to provide a detailed view of the model's performance across true and predicted labels

5.Results & Comparisons

Dataset Version	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC-ROC**
Original Imbalanced	0.9874	0.978723	0.926174	0.951724	0.991197
Vanilla GAN Augmented	0.989637	0.994764	0.984456	0.989583	0.998424
WGAN-GP Augmented	0.9876	0.9856	0.9896	0.9876	0.9979

Original	**Vanilla GAN**	**WGAN**
		



Starting with the original imbalanced dataset, the classifier had a high accuracy (0.9874) and precision (0.9787) for the spam class. But the recall was only 0.9262, meaning it missed a decent number of actual spam messages. This clearly showed the usual problem with imbalanced datasets: the model leaned towards the majority class and struggled to capture the minority class well.

Then I augmented the data using Vanilla GAN, and the impact was immediately noticeable. The recall jumped to 0.9845, and the F1-Score hit 0.9896. Even the overall accuracy slightly increased to 0.9896, and the AUC-ROC improved to 0.9984. So the Vanilla GAN clearly helped the model learn spam patterns better by giving it realistic synthetic samples.

Next, I tried WGAN, and again, the results were super promising. The recall was even higher at 0.9896, showing that WGAN was especially strong at detecting more actual spam messages. The F1-Score was 0.9876, and although the accuracy (0.9876) and AUC-ROC (0.9979) were slightly lower than with Vanilla GAN, the model still performed really well overall. It's clear WGAN-GP produced diverse and high-quality samples that helped balance the classifier's focus across classes.

6. Observations and Conclusions

- GANs improved dataset balance and spam detection performance.
- WGAN achieved the best F1-Score and recall, indicating higher robustness in identifying spam.
- Vanilla GAN also significantly improved classifier performance over the baseline.
- WGAN-GP yielded stronger recall but with minor trade-off in precision.
- Generated texts were generally usable but sometimes repetitive, highlighting the challenge of meaningful text generation

Conclusion: In conclusion, employing GANs for data augmentation is an effective strategy to mitigate the challenges posed by imbalanced datasets in text classification. Both Vanilla GAN and WGAN successfully generated synthetic spam messages that enabled the classifier to learn more comprehensively about the minority class, leading to a more robust and balanced performance.